REMARKS

The comments of the applicant below are each preceded by related comments of the examiner (in small, bold type).

The specification is objected to as failing to provide proper antecedent basis for the claimed subject matter. See 37 CFR 1.75(d)(l) and MPEP 5 608.01 (0). Correction of the following is required: physical article or object.

The applicant disagrees. It is clear from the specification that a physical article or object can be, for example, a processor (specification, page 1, lines 6-8 and 12-13; page 4, line 5) or a physical disk (specification, page 11, line 6).

Claims 1-27, 32-43 and 48-51 are rejected under 35 U.S.C. 103(a) as being unpatentable over Hapner et al. (US. Patent No. 5,727,203) in view of Cheriton (US. Patent No. 5,666,514) and further in view of Somani et al. (US. Patent No. 5,524,212) and further in view of Fleischman (US. Patent No. 6,507,847).

Regarding on claim 1, Hapner teaches a method comprising:

maintaining a database that stores data persistently (col. 7, lines 15-16), accepting tasks from the task sources, at least some of the tasks having competing requirements (multiple threads competing) for user of regions of the database (resources).

Hapner does not explicitly teach data included in a given region not being available for simultaneous access for writing by more than one of the tasks having competing requirements and data included in different regions being available for simultaneous access for writing by more than one of the tasks having competing requirements, assigning each of the region to a available processor, each of the regions being assignable to any of the processors, defining, for each of the tasks, jobs each of which requires write access to regions that are to be accessed by no more than one of the processors and distributing the jobs for concurrent execution by the associated processors.

Hapner does not explicitly discloses assigning each of the region to a available processor, each of the regions being assignable to any of the processors, defining, for each of the tasks, jobs each of which requires write access to regions that are to be accessed by no more than one of the processors and distributing the jobs for concurrent execution by the associated processors.

Claim 1, as amended, recites features related to a persistently maintained database that has been partitioned into regions based on characteristics of data items in the database. Tasks may be in contention to write data to the respective regions. The contention among the tasks is reduced by creating jobs based on each of the tasks, each of the jobs needing to write data in no more than one of the regions. For each of the regions, only those jobs that need to write data in that region are queued; and the queued jobs for each region are executed, one after another and

without contention. The jobs that are queued for different regions are executed simultaneously, in parallel, and without contention.

Hapner's persistent database is not partitioned into regions based on the characteristics of data items. When multiple threads need to access his database, Hapner temporarily locks a piece of code that one thread needs to execute and prevents other threads from accessing that piece of code (column 10, lines 11-22). When a second thread is executed, Hapner temporarily locks another piece of code. Even if the piece of code that is temporarily locked by Hapner could be considered "a region of a database", it is not a region of that is partitioned (in the words of the applicant's claim 1) "based on characteristics of items of data," but instead is determined on the fly at the time a thread needs to access that particular piece of the code. Nor did Hapner create jobs based on each of the tasks, or queue only those jobs that need to write data in a particular region, or execute jobs simultaneously that are queued for different regions.

> **Cheriton discloses data included in a given region not being available for simultaneous access for writing by more than one of the tasks having competing requirements and data included in different regions being available for simultaneous access for writing by more than one of the tasks having competing requirements (...a read operation or an additional shared ownership mode which allows multiple processors and caches to read the data concurrently ...) (col. 6, lines 21-59). This suggests some part of the data cannot be access based on an exclusive lock and other data can be accessed concurrently by plurality of processors. Therefore, it would have been obvious to one ordinary skill in the art at the time of the invention was made to modify teaching Hapner to include some part of the data cannot be access based on a exclusive lock and other data can be accessed concurrently by plurality of processors as disclosed by Cheriton in order to allow the multiple threads to access data.**

The applicant respectfully disagrees. Cheriton described memory cache organizations for recording or logging of memory updates (abstract), but not jobs that need to write data in a persistently maintained database, let alone partitioning such a database into regions "based on characteristics of the data items." In addition, although Cheriton talked about concurrently reading from a cache (column 6, lines 50-54), Cheriton requires an exclusive ownership of a cache line to write data into the cache. When some other process to the cache owns the cache line for writing, the current data writing has to be suspended. Accordingly, Cheriton did not describe and would not have made obvious jobs queued for different regions being executed simultaneously, in parallel, and without contention, as recited by claim 1. Nor did Cheriton

create jobs based on each of the tasks, or queue only those jobs that need to write data in a particular region. Cheriton said:

> When a processor writes a datum to an address, the cache first checks that it has *exclusive* ownership of the associated cache line before allowing the write operation to proceed..... If the processor does not have exclusive ownership, the write operation and the processor are *suspended*, the cache requests exclusive ownership of the cache line from the memory system, loading the cache line if not present in the cache, and then allows the write operation and the processor to proceed once the line is present and its holds exclusive ownership (column 6, lines 24-26 and 28-29).

On the other hand, Somani teaches assigning each of the region to a available processor, each of the regions being assignable to any of the processors, defining, for each of the tasks, jobs each of which requires write access to regions that are to be accessed by no more than one of the processors "as a user writes an INSIGHT program 272 defining application tasks and the number of processor used for each task.. . Each configuration specification file defines a set of generic processors and the jobs partitioned among them" (col. 27, lines 56-62). This teaches associated task with each processor and only one task for that processor. Therefore, it would have been obvious to one ordinary skill in the art at the time of the invention was made to modify Hapner and Cheriton to include the one task for one processor as taught by Somani to allow one processor to read or write the data to the database to keep the data consistency.

Like Cheriton, Somani said nothing about a persistently maintained database or partitioning the database into regions based on characteristics of data items. In addition, Somani did not describe and would not have made obvious reducing contention for writing data to respective regions of a persistently maintained database by "creating jobs that are based on each of the tasks," let alone "queuing only those jobs that need to write data in the region," or "executing jobs simultaneously that are queued for different regions," as recited by claim 1.

Somani assigned jobs among processors for execution (column 27, lines 56-64), based on whether the system was capable of executing the assignment (column 28, lines 2-6). Somani's jobs used external data sources (column 27, line 67 and column 28, line 50-52). However, Somani said nothing about partitioning the data sources, or whether the jobs were in contention to access a region of the data source, or about reducing contention for writing data to respective regions of a persistently maintained database by creating jobs based on each of the tasks.

Furthermore, Fleischman also discloses a large number of threads can be executing in simultaneously within the database. The term "simultaneously" is used in the context known to programmers similar to "multithreading" i.e. multiple threads do not execute in perfectly simultaneous manner unless the server has parallel processor (col. 5, lines 8-20) and multiple small "read" and "write" commands to a disk drive ...(col. 5, lines 33-36).

Fleischman disclose the execution of the multiple write threads, and each of the processor is assign to a specify write to a specific portion of the database. Furthermore, in the write thread no other processor to be able to access to the lock region which is accessed by other processor. Therefore, Fleischman implicitly discloses the concept of assigning the assigning each of the region to a available processor, each of the regions being assignable to any of the processors, defining, for each of the tasks, jobs each of which requires write access to regions that are to be accessed by no more than one of the processors and distributing the jobs for concurrent execution by the associated processors. Therefore, it would have been obvious to one ordinary skill in art at the time of the invention was made to modify both Hapner, Cheriton and Somani system to include executing of the multiple write threads, and each of the processor is assigned to a specify write to a specific portion of the database and no other processor be able to access to the lock region which is accessed by other processor in the write thread as disclosed by Fleischman to resolve any conflict of accessing the database.

Like Hapner, Fleischman did not partition his database into regions "based on characteristics of items of data." Although Fleischman mentioned spinning media (column 5, lines 35-37), he did not describe and would not have made obvious partitioning data stored on the spinning media "based on characteristics of items of data."

Fleischman also did not describe and would not have made obvious "reducing the contention among the received tasks by creating jobs that are based on each of the tasks, each of the jobs needing to write data in no more than one of the regions of a persistently stored database." When Fleischman's server includes only a single processor, he locks the portion of database that a thread was accessing to prevent other threads from interfering (column 5, lines 16-22). When multiple processors are used, different processors could possibly execute different threads in parallel. However, executing threads in parallel by multiple processors does not necessarily imply writing data in different regions of a persistently maintained database in parallel. Fleischman said nothing about his threads being in contention to write data into a region of the database, or about each thread only needing to access no more than one region of his database. Nor did he say anything about "reducing the contention ... by creating jobs that are based on" each thread so that each job only needed to write in no more than one of the regions of the database.

None of Hapner, Cheriton, Somani, or Fleischman, alone or in combination, described or would have made obvious the features of claim 1.

Claim 52 contains similar features to those of claim 1 and is patentable for at least the similar reasons to those discussed with respect to claim 1.

Claims 57-61 are rejected under 35 U.S.C. 103(a) as being unpatentable over Hapner et al. (US. Patent No. 5,727,203) in view of Zaiken et al. (US. Patent No. 5,907,848).

Regarding on claim 57, Hapner teaches a physical article or object bearing instruction to cause a processor to execute a job a job (threads), the job requiring access (write to the database 159) (col. 9, lines 25-30) to a region of a database that stores data persistently (persistence data portion 164) (col. 9, lines 55-60), the job including instruction and pointers to data in the region of the database (cot. 9, lines 55- 60);

Hapner teaches "as corresponding to multiples threads competing for the resource of both the database cache 160 and the persistence database portion 164" (cot. 9, lines 55-60). Hapner does not explicitly teach the job that is executed being selected from a contention space of jobs identified by an index, the jobs in the contention space having competing requirement to write into the region of the database, the index distinguish the contention space from other contention spaces of jobs that do not have competing requirements to write into the region of the database. On the other hand, Zaiken teaches an index that identifies a contention space of jobs that have competing requirement to write into the region of the database, the index distinguish the contention space from other contention spaces of jobs that do not have competing requirements to write into the region of the database "as corresponding to if the file name in the records 20 matches a filename in the selected template 28, the transaction monitor program then searches for any existing index files 30 having job identifier data equal to the job identifier data in the record 20" (cot. 11, lines 39-43). This teaches the index identifying the job in the index files to distinguish by comparing the job identifier in the file. Therefore, it would have been obvious to one ordinary skill in the art at the time of the invention was made to modify Hapner system to include comparing the job identifier in the index files of Zaiken on order to distinguish the jobs by comparing the job identifier in order to process the requested job.

Claim 57 describes an index to identify each queue of jobs that are in contention to write data to a region of a database. The index distinguishes "the queue from other queues of jobs that do not have need to write data into that region of the database." In other words, different indexes identify different queues that are not in contention to write data into a given region of the database.

As the examiner conceded, Hapner did not describe and would not have made obvious such queues of jobs.

Zaiken also did not describe indexes that identify different queues of jobs that are not in contention to write data into a given region of a database. Each of Zaiken's transactions is defined by a template containing names of data files affected by the transaction (abstract). Even if the filenames could be considered indexes to the data files, they are not indexes of different queues of transactions.

At the beginning of each transaction in Zaiken, an index file is created to monitor the progress of the transaction. During the transaction, each data file defined in the template is accessed to complete the transaction. Each time one data file is accessed, the filename is written

into the index file for the transaction. The filenames in the template and those in the index file are compared to determine the progress of the transaction. If all filenames on the template for this particular transaction were in the index file, the transaction is complete and the index file is closed and deleted.

Zaiken's index file is not an index of a queue of jobs, but at most an index of a list of names of data files that one transaction accessed. Even if Zaiken's one transaction were interpreted to have a queue of jobs each accessing one data file identified by one filename, two transactions of Zaiken (i.e., two lists of jobs) may be contending to access in the same data file. For example, one transaction "AA" may have an index file "aa" containing filenames: {A1, B2, D1, G3 ...} and another transaction "BB" may have an index file "bb" containing filenames: {B2, F1, D1, A1, H1, ...}. Jobs identified by indexes "aa" and "bb" are in contention to access files B2, D1, and A1. Accordingly, the index file of Zaiken is not an index that identifies "a queue of jobs" and distinguishes the queue of jobs from "other queues of jobs that do not have need to write data into the region of the database," as recited by claim 57. In addition, as also illustrated in the above example, jobs listed in one index file are not necessarily in contention to access one data file. Thus Zaiken also did not describe and would not have made obvious "the jobs in the queue being in contention to write data to the region of the database", also recited by claim 57.

Zaiken said:

> Each transaction template contains a number of filenames identifying files in the database affected during the transaction defined by the template. (Abstract)
>
> For each record relating to a new transaction, ... a separate index file ... is created and stored for the transaction, ... (column 4, lines 3-7)
>
> The creation of an index file 30 "starts" a new transaction by marking a new transaction as having started at this point in the log 18. (column 11, lines 49-51)
>
> If the filenames from the record and index file do not match, then the record represents another action in the transaction and an entry is added to the index file 30 containing the filename in the record 20 .... (column 12, lines 18-22)
>
> The list of filenames in the index file 30 is then compared to the list of filenames in the selected template 28, step 130. If all the filenames in the selected template 28 are in the index file 30, then the transaction is complete and the index file is therefore closed and deleted, step 132. If not all the filenames in the selected template 28 are in the index file 30, then the transaction may be incomplete, and the next record 20 is retrieved, step 120, and the process continued. (column 12, lies 23-30)

> Once a transaction has started, no other action should be performed on the files involved in the transaction until the transaction is completed or aborted, ..., filename F1 would be locked until transaction 1 is committed or aborted. (column 2, lines 57-65)

Neither Hapner nor Zaiken, alone or in combination, described or would have made obvious the features of claim 57.

> **Claims 68-83 are rejected under 35 U.S.C. 103(a) as being unpatentable over Hapner et al. (US. Patent No. 5,727,203) in view of Fleischman (US. Patent No. 6,507,847 BI).**
>
> **Regarding on claim 68, Hapner teaches a method comprising:**
>
> **Maintaining a database that stores data persistently (col. 7, lines 15-16) and provides a primary level of guarantee that data written in a request transaction is not lost once the transaction is committed (col. 10, lines 11-22), Accepting tasks from the task source for concurrent execution by multiple processors, at least some of the tasks having conflicting requirements to write into the same region of the database (col. 9, lines 54-65), and Hapner teaches, "a mutex is created to corresponding to a piece of code, a portion data, some state, etc ... when a thread has locked a mutex, it is said to "own" the locked mutex. In order for other threads to own the mutex, the first thread (i.e. the thread that locked the mutex) must unlock it. Thus mutexes provide a mechanism by which the programmer can control the serialization of multiple threads, ensuring that steps occur in a desired order and that the state corresponding to the mutex is maintained in a consistent manner" (col. 10, lines 11-22). This teaches the mutex lock is the guarantee that the thread holding the job will be executed and no data is lost.**
>
> **Hapner does not explicitly teaches providing a software mechanism that guarantees, as least to the primary level of guarantee, that the tasks will be executed without loss of data and without occurrence of any actual conflict with respect to the region of the database. Further more, Fleischman discloses providing a software mechanism that guarantees, as least to the primary level of guarantee, that the tasks will be executed without loss of data and without occurrence of any actual conflict with respect to the region of the database (a large number of threads can be executing in simultaneously within the database. The term "simultaneously" is used in the context known to programmers similar to "multithreading" i.e. multiple threads do not execute in perfectly simultaneous manner unless the server has parallel processor (col. 5, lines 8-20) and multiple small "read" and "write" commands to a disk drive ...(col. 5, lines 33-36).**
>
> **Fleischman disclose the execution of the multiple write threads, and each of the processor is assign to a specify write to a specific portion of the database.**
>
> **Furthermore, in the write thread no other processor to be able to access to the lock region which is accessed by other processor. Therefore, Fleischman implicitly discloses the concept of assigning the assigning each of the region to a available processor, each of the regions being assignable to any of the processors, defining, for each of the tasks, jobs each of which requires write access to regions that are to be accessed by no more than one of the processors and distributing the jobs for concurrent execution by the associated processors These multiple write thread execution in concurrently will guarantee that tasks will be executed without the loss of data and without occurrence of any actual conflict with respect to the region of the database. Therefore, it would have been obvious to one ordinary skill in the art at the time of the invention was made to modify Hapner system to include multiple write threads concurrently execution with different portions of data as taught by Fleischman in order guarantee that all tasks will be executed without conflict and loss of data.**

Claim 68 recites translating each task to be executed in a database into jobs, each job needing to write data in no more than one of the partitioned regions of the database.

The examiner conceded that Hapner did not describe and would not have made obvious the features of claim 68.

Fleischman, as explained with respect to claim 1, did not partition his database into regions based on characteristics of items of data. Nor did Fleischman translate each task into jobs, each job needing to write data in no more than one of the regions. Although Fleischman described possible parallel execution of threads by multiple processors, contrary to the examiner's assertion, Fleischman said nothing about **"each of the processor is assign to a specify write to a specific portion of the database"**. Fleischman only said that multiple threads may be searching the database, and that in spinning media (column 5, lines 23-25), blocks of data can be processed in adjacent portions of storage (column 5, lines 35-37). However, Fleischman did not translate each task into jobs, each job needing to write data in no more than one of the regions.

Accordingly, neither Hapner nor Fleischman, alone or in combination, described or would have made obvious the features of claim 68.

Claims 28-31 and 44-47 are rejected under 35 U.S.C. 103(a) as being unpatentable over Hapner et al. (US. Patent No. 5,727,203) in view of Cheriton in view of (US. Patent No. 5,666,514) and further in view of Somani et al. (US. Patent No. 5,524,212) and further in view of Fleischman (US. Patent No. 6,507,847 B) and further in view of Murray (US. Patent No. 5,404,521).

Claims 28-31 and 44-47 are patentable for at least the reasons discussed with respect to claim 1, from which they depend.

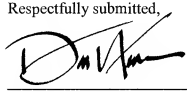
All of the dependent claims are patentable for at least the reasons for which the claims on which they depend are patentable.

Canceled claims, if any, have been canceled without prejudice or disclaimer.

Any circumstance in which the applicant has addressed certain comments of the examiner does not mean that the applicant concedes other comments of the examiner. Any circumstance in which the applicant has made arguments for the patentability of some claims does not mean that there are not other good reasons for patentability of those claims and other claims. Any circumstance in which the applicant has amended or canceled a claim does not mean that the applicant concedes any of the examiner's positions with respect to that claim or other claims.

Please apply $555 for the Petition for Extension of Time fee and any other charges or credits to deposit account 06-1050, referencing attorney docket 11811-0008002.

Respectfully submitted,



Date: 2/20/9

David L. Feigenbaum
Reg. No. 30,378

Fish & Richardson P.C.
225 Franklin Street
Boston, MA 02110
Telephone: (617) 542-5070
Facsimile: (877) 769-7945

22135847.doc